



ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ
ΥΠΟΛΟΓΙΣΤΩΝ

ΔΕΥΤΕΡΗ ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΝΑΦΟΡΑ
ΛΕΙΤΟΥΡΓΙΚΑ ΣΥΣΤΗΜΑΤΑ

Ομάδα 25
Αναστασία Χριστίνα Λίβα
03119029
Γεώργιος Μυστριώτης
03119065

Περιεχόμενα

Άσκηση 1	2
Άσκηση 2	7
Άσκηση 3	12
Άσκηση 4	17

Άσκηση 1

Ακολουθεί ο πηγαίος κώδικας για την πρώτη άσκηση

```
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <signal.h>
#include <assert.h>

#include <sys/wait.h>

#include "proc-common.h"

#define SLEEP_SEC 10

void fork_procsc();

void fork_procsd(){
    pid_t pid;
    int status;
    pid=fork();
    if (pid < 0) {
        perror("fork");
        exit(1);
    }
    if (pid == 0) {
        /* Child */
        printf("hello, process D here!\n");
        printf("process D sleeping\n");
        sleep(SLEEP_SEC);
        printf("process D terminated\n");
        exit(13);
    }
    else {
        printf("process B waiting for process D\n");
        //printf("D - %d\n",pid);
        pid=wait(&status);
        explain_wait_status(pid, status);
    }
}
```

```
    }  
  
}  
  
void fork_procsb() {  
    pid_t pid;  
    int status;  
  
    pid=fork();  
    if (pid < 0) {  
        perror("fork");  
        exit(1);  
    }  
    if (pid == 0) {  
        /* Child */  
        printf("hello, process B here!\n");  
        fork_procsd();  
        printf("process B terminated\n");  
        exit (19);  
    }  
    else {  
        printf("process A waiting for  
        process B\n");  
        //printf("B - %d\n", pid);  
        fork_procsc();  
        pid=wait(&status);  
        explain_wait_status(pid, status);  
        //fork_procsc();  
    }  
  
}  
  
}  
void fork_procsc () {  
    pid_t pid;  
    pid=fork();  
    int status;
```

```
        if (pid < 0) {
            perror("fork");
            exit(1);
        }
        if (pid == 0) {
            /* Child */
            printf("hello, process C here!\n");
            printf("process C sleeping\n");
            sleep(SLEEP_SEC);
            printf("process C terminated\n");
            exit(17);
        }
        else {
            printf("process A waiting for process C\n");
            //printf("C - %d\n", pid);
            pid=wait(&status);
            explain_wait_status(pid, status);
        }
    }
}

int main(void)
{
    pid_t pid, mpid;
    int status;
    /* Fork root of process tree */
    pid = fork();
    if (pid < 0) {
        perror("main: fork");
        exit(1);
    }
    if (pid == 0) {
        printf("hello, process A here!\n");
        /* Child */
        mpid=getpid();
        fork_procsb();
        printf("process A terminated\n");
    }
}
```

```
        exit(16);  
  
    }  
    else {  
        mpid=pid;  
        show_pstree(mpid);  
        pid = wait(&status);  
        explain_wait_status(pid, status);  
    }  
    return 0;  
}
```

Τρέχοντας τον κώδικα αυτό παίρνουμε το εξής αποτέλεσμα:

```
oslaba25@os-nodel:~$ ./ask2a  
hello, process A here!  
process A waiting for process B  
hello, process B here!  
process A waiting for process C  
hello, process C here!  
process C sleeping  
process B waiting for process D  
hello, process D here!  
process D sleeping  
  
ask2a(13061)└─ask2a(13063)──ask2a(13065)  
               └─ask2a(13064)  
  
process C terminated  
process D terminated  
My PID = 13061: Child PID = 13064 terminated normal  
  
ly, exit status = 17  
My PID = 13063: Child PID = 13065 terminated normal  
  
ly, exit status = 13  
process B terminated  
My PID = 13061: Child PID = 13063 terminated normal  
  
ly, exit status = 19  
process A terminated  
My PID = 13060: Child PID = 13061 terminated normal  
  
ly, exit status = 16
```

Ερώτηση Πρώτη: Σε περίπτωση που η διεργασία A τερματιστεί πρόωρα, δηλαδή πριν τερματιστούν όλα τα παιδιά της, με την εντολή `kill -KILL <pid>` τότε τα παιδιά που δεν έχουν τερματιστεί ακόμα «υιοθετούνται» από την διεργασία `init`. Αφού η A τερματίζεται πρόωρα μέσω σήματος δε λαμβάνουμε μήνυμα από την `explain_wait_status` για τις διεργασίες παιδιά της A. Η διεργασία `init` αποτελεί την πρώτη διεργασία που ξεκινά από την εκκίνηση του συστήματος και αναμένει τον τερματισμό των ορφανών παιδιών.

Ερώτηση Δεύτερη: Όταν μια διεργασία χρησιμοποιεί την έντολη `getpid()` επιστρέφεται η `pid` του εαυτού της. Συνεπώς με την εντολή `show_pstree(getpid())` η ρίζα του δέντρου μας θα είναι ο πατέρας της διεργασίας A. Τελικώς θα προκύψει το ίδιο δέντρο διεργασιών που προέκυψε νωρίτερα με επιπλέον κόμβο τη διεργασία πατέρα της A ως ρίζα. Ακόμα, εμφανίζονται ως παιδιά της ρίζας οι διεργασίες `sh` και `ps tree` που καλούνται από την `show_pstree`.

Ερώτηση Τρίτη: Το όριο αυτό υπάρχει ώστε να μην υπερφορτωθεί το σύστημα εάν κάποιος χρήστης εκτελέσει πάρα πολλά `forks` εξαντλώντας έτσι όλους τους πόρους του συστήματος.

Άσκηση 2

```
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <signal.h>
#include <assert.h>

#include <sys/wait.h>

#include "proc-common.h"
#include "tree.h"
#define SLEEP_SEC 10

void traverse(struct tree_node* node, int current_child){
    pid_t pid;
    int status, j;

    printf("Process %c created with ID = %d\n",
        node->name[0], getpid());

    if(node->nr_children == 0){
        sleep(SLEEP_SEC);
        printf("Process %c exiting\n",
            node->name[0]);
        exit(0);
    }
    struct tree_node* cp;
    cp=node->children + current_child - 1;
    pid_t PID[node->nr_children];
    if(node->nr_children != 0){
        for (j=0; j<node->nr_children; j++){
            PID[j] = fork();
            if (PID[j] < 0) {
                perror("fork");
                exit(1);
            }
            else if (PID[j] == 0) {
                traverse(cp, j);
            }
            else{

```



```
        //pid=wait (&status);
    }
    cp++;
}
for(j=0; j<node->nr_children; j++){
    PID[j]=wait (&status);
    explain_wait_status(PID[j],
        status);
}
exit(0);
return;
}

int main(int argc, char* argv[]){
    pid_t pid;
    int status;
    if (argc!=2)
        printf("Wrong usage of command.\n");

    struct tree_node *root;
    root = get_tree_from_file(argv[1]);

    pid=fork();
    if (pid < 0) {
        perror("main: fork");
        exit(1);
    }
    else if (pid == 0) {
        //printf("hello, root here!\n");
        traverse(root, 1);
        exit(16);
    }
    else {
        show_pstree(pid);
        pid=wait (&status);
        explain_wait_status(pid, status);
    }
    return 0;
}
```

Ακολουθούν τα αποτελέσματα του κώδικα για τα δέντρα που περιλαμβάνονται στο

tree1.txt και tree2.txt αντίστοιχα.

```
oslab25@os-node1:~$ ./prox kat1.txt
Process A created with ID = 13316
Process B created with ID = 13318
Process C created with ID = 13319
Process D created with ID = 13320
Process E created with ID = 13321

prox(13316)---prox(13318)---prox(13320)
              |               |
              |               |---prox(13321)
              |---prox(13319)

Process C exiting
Process D exiting
My PID = 13316: Child PID = 13319 terminated normal
ly, exit status = 0
Process E exiting
My PID = 13318: Child PID = 13320 terminated normal
ly, exit status = 0
My PID = 13318: Child PID = 13321 terminated normal
ly, exit status = 0
My PID = 13316: Child PID = 13318 terminated normal
ly, exit status = 0
My PID = 13315: Child PID = 13316 terminated normal
ly, exit status = 0
```


Ερώτηση Πρώτη: Τα μηνύματα έναρξης εμφανίζονται κατά πλάτος του δέντρου, πράγμα λογικό καθώς κάθε διεργασία γονιός δημιουργεί τις διεργασίες παιδιά του διαδοχικά στο `for loop`. Τερματίζουν με αντίστροφο τρόπο ξεκινώντας από τα φύλλα (χωρίς συγκεκριμένη σειρά ανάμεσα στις διεργασίες που βρίσκονται στο ίδιο επίπεδο του δέντρου) καθώς κάθε γονέας πρέπει να περιμένει τα παιδιά του να τερματίσουν πρώτου τερματίσει ο ίδιος.

Άσκηση 3

```
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include <signal.h>
#include <sys/types.h>
#include <sys/wait.h>

#include "tree.h"
#include "proc-common.h"
#define SLEEP_SEC 3

void traverse(struct tree_node* node,
int current_child){
    pid_t PID[node->nr_children];
    pid_t pid;
    int status;

    printf("%c - %d\n", node->name[0], getpid());

    if(node->nr_children == 0) {
        raise(SIGSTOP);
        printf("PID = %ld, name = %s is awake\n",
(long)getpid(), node->name);
        exit(0);
    }
    if(node->nr_children != 0){
        struct tree_node* cp;
        cp=node->children + current_child - 1;
        int i;
        for(i=0;i<node->nr_children;i++){
            pid = fork();
            if (pid < 0) {
                perror("fork");
                exit(1);
            }
            else if (pid == 0) {
                traverse(cp,1);
            }
        }
    }
}
```

```
        else{

            PID[i]=pid;
        }
        cp++;
    }
    wait_for_ready_children(node->nr_children);
    raise(SIGSTOP);
    printf("PID = %ld, name = %s is awake\n",
        (long)getpid(), node->name);
    for (i=0;i<node->nr_children;i++){
        //printf("%d\n",i);
        kill(PID[i], SIGCONT);
        PID[i]=wait(&status);
        explain_wait_status(PID[i], status);
    }
    exit(0);
}

return;

}

int main(int argc, char *argv[]){
    pid_t pid;
    int status;
    struct tree_node *root;

    if (argc < 2){
        fprintf(stderr, "Usage: %s <tree_file>\n",
            argv[0]);
        exit(1);
    }

    /* Read tree into memory */
    root = get_tree_from_file(argv[1]);

    /* Fork root of process tree */
    pid = fork();
```

```
if (pid < 0) {
    perror("main: fork");
    exit(1);
}
if (pid == 0) {
    /* Child */
    traverse(root, 1);
}
else{
    wait_for_ready_children(1);
    show_pstree(pid);
    //printf("%d\n", pid);
    kill(pid, SIGCONT);
    wait(&status);
    explain_wait_status(pid, status);
}
return 0;
}
```

Τρέχοντας τον παραπάνω κώδικα χρησιμοποιώντας το αρχείο `tree.txt` παίρνουμε το εξής αποτέλεσμα:

```
A - 13958
B - 13959
C - 13960
My PID = 13958: Child PID = 13960 has been stopped

by a signal, signo = 19
D - 13961
My PID = 13959: Child PID = 13961 has been stopped

by a signal, signo = 19
E - 13964
My PID = 13959: Child PID = 13964 has been stopped

by a signal, signo = 19
My PID = 13958: Child PID = 13959 has been stopped

by a signal, signo = 19
My PID = 13957: Child PID = 13958 has been stopped

by a signal, signo = 19

ask2c(13958)└─ask2c(13959)└─ask2c(13961)
               └─ask2c(13964)
               └─ask2c(13960)

PID = 13958, name = A is awake
PID = 13959, name = B is awake
PID = 13961, name = D is awake
My PID = 13959: Child PID = 13961 terminated normal

ly, exit status = 0
PID = 13964, name = E is awake
My PID = 13959: Child PID = 13964 terminated normal

ly, exit status = 0
My PID = 13958: Child PID = 13959 terminated normal

ly, exit status = 0
PID = 13960, name = C is awake
My PID = 13958: Child PID = 13960 terminated normal

ly, exit status = 0
My PID = 13957: Child PID = 13958 terminated normal

ly, exit status = 0
```


Ερώτηση Πρώτη: Χρησιμοποιώντας τα σήματα δεν χρειάζεται να περιμένουμε τον τερματισμό των διεργασιών μέσω του `sleep` καθώς είναι εφικτή η αλλαγή κατάστασης της κάθε διεργασίας όποτε αυτή είναι επιθυμητή άρα εξοικονομείται χρόνος αφού αποφεύγεται η εντολή `sleep`. Επίσης με τα σήματα έχουμε επικοινωνία μεταξύ των διεργασιών. Επιτυγχάνεται συγχρόνως η εμφάνιση μηνυμάτων κατά DFS, ενώ πλέον είναι βέβαιη η ορθή λειτουργία της `show_pstree`.

Ερώτηση Δεύτερη: Η χρήση του `wait_for_ready_children` στην περίπτωση αυτή εξασφαλίζει ορθή διάσχιση κατά βάθος του δέντρου διεργασιών αφού κάθε διεργασία πατέρας περιμένει να είναι έτοιμες οι διεργασίες παιδιά του για να τεθεί σε κατάσταση παύσης στέλνοντας στον εαυτό του μέσω του `raise` το σήμα `SIGSTOP`.

Άσκηση 4

```
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <signal.h>
#include <assert.h>

#include <sys/wait.h>

#include "proc-common.h"
#include "tree.h"
#define SLEEP_SEC 10

void traverse(struct tree_node* node, int *pfdback){
    pid_t pid;
    int status;
    int value;
    //printf("%c\n", node->name[0]);
    //printf("%c - %d\n", node->name[0], getpid());

    if(node->nr_children == 0){
        value=atoi(node->name);
        if (write(pfdback[1], &value,
        sizeof(value)) != sizeof(value)) {
            perror("write to pipe");
            exit(1);
        }
        printf("I am returning value = %d to my parent\n",
        value);
        close(pfdback[1]);
    }
    if(node->nr_children !=0){
        struct tree_node* cp;
        cp=node->children;
        int num[node->nr_children];
        int i;
        for(i=0;i<node->nr_children;i++){
            int pfd[2];
            num[i]=0;
```

```
if (pipe(pfd) < 0) {
    perror("pipe");
    exit(1);
}

pid = fork();
if (pid < 0) {
    perror("fork");
    exit(1);
}
else if (pid == 0) {
    traverse(cp, pfd);
}
else{
    pid=wait(&status);
    //explain_wait_status(pid, status);
    if (read(pfd[0], &value,
        sizeof(value)) != sizeof(value)) {
        perror("read from pipe");
        exit(1);
    }
    num[i]=value;
    close(pfd[0]);

    }

    cp++;
}
//printf("%d %d\n", num[0], num[1]);
if (node->name[0]=='+') {
    value=num[0]+num[1];
}
else if (node->name[0]=='*') {
    value=num[0]*num[1];
}
printf("Performing %d %c %d\n",
num[0], node->name[0], num[1]);
if (write(pfdback[1], &value,
    sizeof(value)) != sizeof(value)) {
    perror("write to pipe");
    exit(1);
}
```

```
        }
        printf("I am returning value = %d to
        my parent\n", value);
    }
    close(pfdback[1]);
    exit(0);
    return ;
}

int main(int argc, char* argv[]){
    pid_t pid;
    if (argc!=2)
        printf("Wrong usage of command.\n");

    struct tree_node *root;
    root = get_tree_from_file(argv[1]);

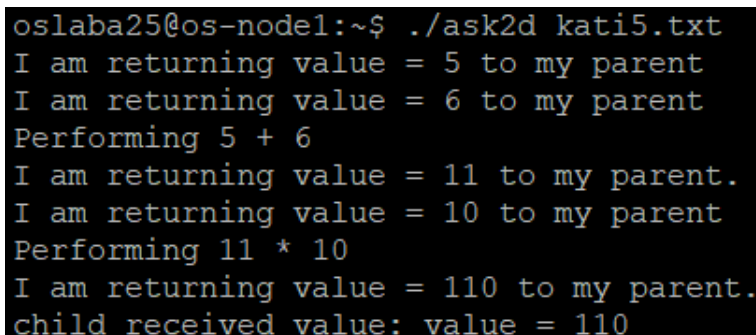
    int status;
    int value;
    int pfd[2];

    if (pipe(pfd) < 0) {
        perror("pipe");
        exit(1);
    }

    pid=fork();
    if (pid < 0) {
        perror("main: fork");
        exit(1);
    }
    else if (pid == 0) {
        traverse(root, pfd);
        exit(0);
    }
    else {
        pid=wait(&status);
        if (read(pfd[0], &value,
        sizeof(value)) != sizeof(value)) {
            perror("read from pipe");
        }
    }
}
```

```
        exit(1);
    }
    printf("child received value:
value = %d\n", value);
    close(pfd[0]);
}
return 0;
}
```

Τρέχοντας τον κώδικα με το calculate.txt παίρνουμε:



```
oslaba25@os-node1:~$ ./ask2d kati5.txt
I am returning value = 5 to my parent
I am returning value = 6 to my parent
Performing 5 + 6
I am returning value = 11 to my parent.
I am returning value = 10 to my parent
Performing 11 * 10
I am returning value = 110 to my parent.
child received value: value = 110
```

Ερώτηση Πρώτη: Ανά διεργασία θέλουμε 2 pipes, εκτός εάν πρόκειται για φύλλα όπου θέλουμε ένα. Για κάθε γονέα και τα αντίστοιχα δύο παιδιά του μπορούμε να δουλέψουμε με ένα pipe, καθώς ο πολλαπλασιασμός και η πρόσθεση χαρακτηρίζονται από την αντιμεταθετική ιδιότητα ως πράξεις. Κατά συνέπεια, δεν υφίσταται διαφορά στο αποτέλεσμα εάν οι τιμές των παιδιών ληφθούν με διαφορετική σειρά. Για κάθε αριθμητικό τελεστή είναι αδύνατη η χρήση μιας σωλήνωσης αφού για παράδειγμα στην αφαίρεση πρέπει να γνωρίζουμε ποια τιμή αφαιρείται από ποια.

Ερώτηση Δεύτερη: Αναθέτοντας κάθε διεργασία σε διαφορετικό πυρήνα παίρνουμε το αποτέλεσμα ταχύτερα αφού ο υπολογισμός μοιράζεται σε πολλούς επεξεργαστές που δουλεύουν συγχρόνως.¹

¹Όλα τα txt αρχεία που αναφέρονται σε αυτό το pdf βρίσκονται στο αρχείο zip που υποβλήθηκε.